Chapter 4
Principles Ensuring Anti-fragility

While it is impossible to predict all potential swan events that can severely impact complex information and communications technology (ICT) systems, we know the general reasons for extreme global behavior: single points of failure such as shared resources, local failures initiating systemic failures due to chain reactions, scaling effects, and cascading failures between system layers and different systems [35, Chap. 4]. Because the removal of single points of failure is a well-understood problem [47], this chapter first introduces four design principles that together isolate local failures before they propagate and cause systemic failures. It then presents one operational principle to quickly remove exploitable vulnerabilities. Finally, the chapter discusses how a systemic failure can occur in a complex adaptive system even when no parts fail, as well as the need to build models to understand such extreme global behavior.

The current chapter defines and illustrates five principles needed to design and operate anti-fragile ICT systems, while the following chapters discuss how these principles can be implemented in different types of complex ICT systems. The principles are rooted in the analysis in Chap. 2, showing the need to isolate local failures and use natural and induced failures to learn about vulnerabilities. The reader may recognize some of the principles as software patterns described in books on software design [35, 48]. Here, we use the term *principle* rather than *pattern* to emphasize that these ideas or concepts can be found in many research fields, not only software design [3, 4, 19, 35, 48, 49, 50].

4.1 Modularity

A complex adaptive ICT system with tightly interconnected units tends to exhibit surprising and undesirable global behavior due to the many non-linear interactions between the units [5, 6]. A local failure due to an internal error in a unit or abnormal

K.J. Hole, *Anti-fragile ICT Systems*, Simula SpringerBriefs on Computing 1,
DOI 10.1007/978-3-319-30070-2_4

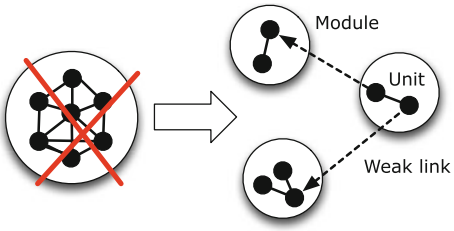

Fig. 4.1 Because a system of tightly interconnected units facilitates systemic failures, we need a system of modules with weak links (*dashed lines*) that break when modules experience local failures

interactions between several units could cause problems for other units and eventually take down the whole system. The first step to avoid propagating local failures in networked computer systems is to modularize the systems at both the hardware and software levels [1, 3, 4]. Conceptually, we represent the modules of a system by nodes in a hierarchical system graph (Figure 4.1 illustrates one level of the graph), where each module is a subgraph of tightly cohesive units.

We use the expressions *strong connection* and *weak connection* to describe the varying levels of dependence between system modules. The terms *dependency* and *connection* are used in much the same way in this book. A module $\mathcal{A}$ is *strongly connected* with (or strongly dependent on) a module $\mathcal{B}$ if $\mathcal{A}$'s functionality is badly affected when $\mathcal{B}$ misbehaves or fails. The module $\mathcal{A}$ is *weakly connected* (or weakly dependent) if $\mathcal{A}$'s important functionality is preserved when $\mathcal{B}$ malfunctions or terminates. When modules are weakly connected, a change to a module should not necessitate changes to any other module. The modules must have well-defined interfaces and these interfaces must be the only way modules can interact with each other. In particular, the internal state of a module must not be directly accessible to another module, but only made available via an interaction mechanism that communicates state information. A communication protocol is an important example of an interaction mechanism.

The system graphs in Fig. 4.1 illustrate the transition from a system of tightly interconnected units to a system of weakly connected modules. The units constituting a module depend on the system level being studied. If we study a complete software solution consisting of a set of well-defined software services, then a module is a service and a unit is a collection of subroutines. In a distributed hardware system, for example, a collection of network routers, a printed circuit board is a unit, while a module is a collection of boards that constitute a cohesive part of a hardware device. If we study interconnected systems, then a module is a whole system.

It is important to understand the difference between strong and weak dependencies in modular systems. Strong dependencies were actually first defined in Sect. 2.7, although the definition did not explicitly introduce the concept of strength. The same section stated that the impact of recurrent incidents in a modular system can be mitigated by introducing additional strong dependencies between the modules. Unfortunately, we may introduce new positive feedback loops at the same time, thus

increasing the probability of extreme global behavior in the form of nonrecurrent swan incidents. The next section outlines how to avoid swans by limiting the strength of dependencies in modular systems.

4.2 Weak Links

When the functionality of a module $\mathcal{A}$ at some system level depends on the functionality of another module $\mathcal{B}$, there is a directed link from $\mathcal{A}$ to $\mathcal{B}$ in the system graph to represent this dependency. In Fig. 4.1, each directed link signifies the relation *depends on*. Different dependencies have varying strengths [5]. We can measure the strength of a dependency by determining the damage a misbehaving module causes in the dependent module.

The next step to prevent local failures from propagating is to ensure that the incoming links to a misbehaving module break in such a way that there is little or no damage to the dependent modules. These so-called *weak links* [49] enhance robustness to propagating failures by restricting damage to a single module. The weak links are represented by dashed lines in Fig. 4.1.

A weak link can be compared to a circuit breaker that protects an electrical system against excessive current. The circuit breaker is an automatic electrical switch designed to detect a fault condition and interrupt current flow. Unlike a fuse, which operates once and then must be replaced, a circuit breaker can be reset to resume normal operation. We are interested in weak links that can restore themselves after they break. Chapter 5 studies how to implement weak links with default fallback responses.

It is necessary to determine the dependencies between modules at different levels of a system [2]. Modules are weakly connected when they have weak links. If the hierarchical system graph of weakly connected modules (see Fig. 4.1) is sparse and of limited size, then the remaining fragility can be analyzed. A dense and large graph of strong dependencies signals intolerable fragility because it becomes hard to determine the cause(s) of an incident and, therefore, countermeasures to avoid similar incidents in the future [4].

4.3 Redundancy

According to Taleb [10], redundancy is an inherent property of anti-fragile systems. They do not make "efficiency" their primary goal. Since the goal of anti-fragile systems is to thrive in randomness, the systems contain "inefficiencies" through layered redundancies. Computer systems enhance their robustness to module failures by

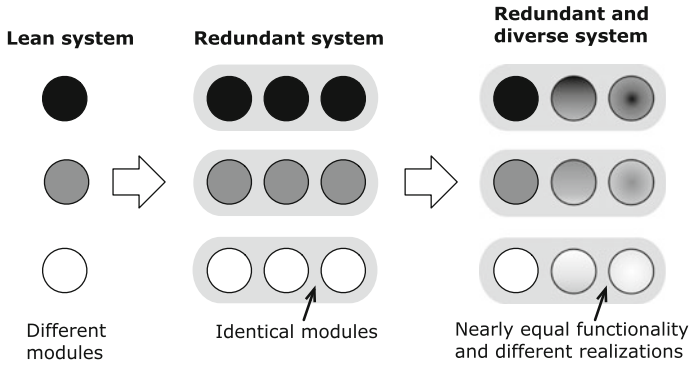

Fig. 4.2 Transition from a lean system to a redundant system and then to a system with both redundancy and diversity

deploying multiple copies of modules at the software and hardware levels. Figure 4.2 illustrates the transition from a lean system to a redundant system. The redundancy is obtained by introducing extra copies of each module.

Two examples illustrate the redundancy principle. First, when a virtual machine fails in a cloud-based system, an identical instance is started automatically. Second, a critically important system should have at least one secondary backup system that runs in parallel with the primary system to ensure a safe fallback. Leading up to the next principle, we note that the secondary system should differ from the primary system to avoid both failing for the same reasons.

4.4 Diversity

A modular system has diversity [50] when it contains differently designed or implemented modules with (nearly) the same functionality. Figure 4.2 depicts the transition from a redundant system to a system that is both redundant and diverse. Diversity makes it less likely that many modules will fail at the same time. Only a diverse system is highly robust to propagating failures; single modules remain fragile. Failures of fragile modules are warning signals of impending systemic instability. If a computer system is a "monoculture," where all computing devices are based on the same hardware or run the same software [28, 29], then it is highly fragile, because a local failure can propagate very easily. This is particularly true for infectious malware that can easily spread to many modules in a large software monoculture. The use of software diversity to halt malware spreading is discussed in Part III.

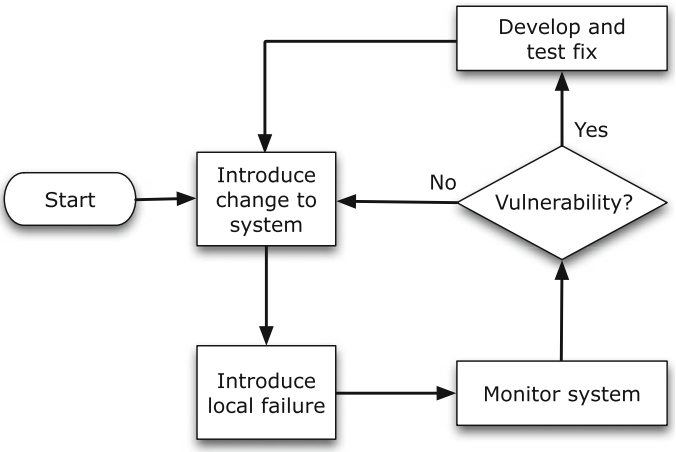

Fig. 4.3 How to use the fail fast principle in an ICT system

4.5 Fail Fast

To create complex adaptive systems that are anti-fragile to classes of negative events, it is necessary to learn from problems and downright failures in the systems because it is effectively impossible to predict all future incidents with a large negative impact. Hence, a system should fail early when the impact is small and stakeholders should learn from these incidents how to adapt the system to limit the impact of future incidents.

When the four design principles of modularity, weak links, redundancy, and diversity are used to avoid failure propagation, we can induce local failures (with only a tiny probability of systemic failure) to detect vulnerabilities early and quickly learn how to improve the ability to prevent propagating failures. The flow diagram in Fig. 4.3 illustrates how the fail fast principle can be used in a system. Netflix pioneered the depicted technique in its cloud-based subscription service for films and TV series (http://techblog.netflix.com). Chapter 5 will discuss Netflix's realizations of the operational fail fast principle and the four outlined design principles.

4.6 Systemic Failure Without Failed Modules

A local failure can propagate over a system and cause a systemic failure. Although there is a strong tendency to assume that a local failure is a well-defined event occurring inside a single module, this is not necessarily true for complex adaptive systems. A well-functioning technical system with normally behaving stakeholders could drift into a systemic failure in the form of a swan event without any well-defined initial

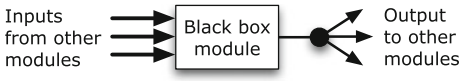

Fig. 4.4 A module in a complex adaptive system viewed as a black box with multiple inputs from other modules and a single output connected to yet other modules

module failure. Swans are often caused by internal and external changes that affect the global pattern of interactions between the modules, between the stakeholders, and between the stakeholders and modules. The changes all seem reasonable when studied in isolation. It is only the combination of the changes that causes a systemic failure [18].

To better understand how a systemic failure can occur without any module failure, we consider a module as a black box with multiple input links and a single output link (see Fig. 4.4). A module receives inputs from other modules and generates an output that becomes input to yet other modules. A module is designed to generate particular output values from combinations of specified input values. If a module receives an unknown or a partial combination of inputs that it was not designed to handle, it can produce an extreme output value. When the extreme output becomes input to another module, it can result in another extreme output. Hence, an unusual combination of inputs to a module can cause more and more modules to generate extreme outputs, leading to a systemic failure. This rare and extreme global behavior occurs despite all modules correctly executing their designed input–output transformations, that is, there are no module failures per se.

The reader should note that incomplete or extreme input combinations to modules could occur due to random noise or temporary faults in the communication links. These transient failures can be hard to recreate and may even be completely overlooked during an investigation to determine why a system misbehaved. This is particularly true when multiple transient errors combine to create incomplete or extreme input combinations.

Since classical risk analysis is based on the notion that a large failure is caused by a chain of smaller events initiated by a well-defined starting event, the analysis may not predict the above systemic failure. The classical approach to risk analysis based on simple, or linear, cause and effect thinking only works satisfactorily when the system's parts are weakly connected with limited interaction. Complex adaptive systems are often strongly connected with a great deal of interaction. To understand the risks associated with complex systems, analysts must avoid thinking that restricts failures to simple chains of events, because this approach ignores potential swans and thus seriously underestimates the total risk taken by stakeholders [18].

The earlier stated principle of weak links is critical to avoid failure propagation that is not initiated by a local failure in a single module but caused by an unusual combination of inputs to one or more modules leading to extreme global behavior. Consider a system monitoring its modules to break the output links, perhaps after

some delay, when the modules produce extreme output. The modules exist in a (logical) hierarchy, where each module belongs to a particular level. A given module may receive inputs from several modules at a lower level. Even if each of the lower-level modules generates normal output, the combination of values taken as input to the upper-level module may still cause this module to generate extreme output. However, since the module is monitored and stopped when it generates extreme output, a systemic failure is, most likely, avoided.

4.7 The Need for Models

While it is quite easy to understand the descriptions of the five principles, it is hard to determine how to realize them in complex adaptive ICT systems to achieve anti-fragility to a particular type of impact. Paraphrasing Yaneer Bar-Yam [51], we argue that it is necessary to create system models, especially during the design phase, to ensure anti-fragility.

The beginning of Chap. 1 discussed the complexity of an ICT system consisting of a large networked computer system and many stakeholders (see Fig. 1.1). The complexity is due to the numerous interactions between the stakeholders and the computer system, the large amounts of communications between the networked subsystems, and the influence of changing security and privacy policies, as well as threats such as equipment failure, extreme weather, and sabotage. An alternative to this communication view of complexity is the behavioral complexity obtained by viewing a complete ICT system as a black box and then studying the minimum amount of information, measured in bits, needed to describe all possible input-output relations.

Let us consider an ICT system with N_{in} input values and N_{out} output values. The values can be in the form of vector or scalar values. We need a minimum of $A = \log_2 N_{out}$ bits to represent an output because all the 2^A outputs must have unique descriptions. Similarly, we need $I = \log_2 N_{in}$ bits to uniquely label an input. The labels allow us to order the inputs. Assume that we have an ordered list of 2^I entries, where the first entry contains the output corresponding to the first input, the second entry contains the output corresponding to the second input, and so on. Since we need A bits to specify an output, the total number of bits needed to completely describe all input–output relations is $2^I \cdot A$. This expression measures the behavioral complexity of an ICT system.

The idea of classical software development is to build a system that realizes a set of well-defined input–output relations. Before the system goes into production, it must be tested. A complex adaptive ICT system with huge numbers of computational devices and users has a huge number of possible inputs. If, for example, $I = 200$ bits, then the complexity is greater than $2^{200} \approx 10^{60}$ bits, which is an enormous number.

Since it is clearly impossible to exhaustively test all inputs, theory is essential to understand how to realize the five principles in complex adaptive systems. Models are especially useful because they characterize global emergent behaviors without having to test all possible inputs. For systems without adequate models, the limitations of testing lead to significant uncertainty about the systems' global behaviors, especially their fragility to swans. While models help reduce the risk to stakeholders, complex adaptive systems will always have hidden risks due to their highly non-linear and time-varying relations between the inputs and outputs [7]. Hence, as first stated in Sect. 2.5, there is no absolute guarantee that complex ICT systems are swan free.

In Chap. 3, we built a model to understand how a user population's trust in an ICT system could change from pervasive trust to massive distrust. Because of the great behavioral complexity, no effort was made to accurately model all aspects of the trust relationship between users and system operators. Instead, we developed an explanatory toy model. Although toy models cannot predict the detailed behavior of systems, the models can be used to uncover fragility to particular types of impacts.

4.8 Discussion

The four design principles of *modularity*, *weak links*, *redundancy*, and *diversity* and the *fail fast* operational principle are not new, since various descriptions can be found in different research fields [3, 4, 19, 35, 48, 49, 50]. However, Taleb's [8, 9, 10] conceptual foundation and the way the principles are melded in Part II outline a novel strategy to design and operate anti-fragile ICT systems.

The reader may wonder if the five principles are sufficient to ensure anti-fragility to any given class of impacts. At the time of this writing, in late 2015, the answer to this question is not fully known. Most likely, the set of principles needed to design and operate an anti-fragile system depends on the type of system and the class of impacts considered. In Parts II and III, we argue that the five principles provide anti-fragility to downtime and malware spreading. More work is required to determine the need for additional principles. A short discussion of possible additional design principles can be found in Chap. 13.

Chapter 3 argued that it is important to build trust between the owner and the users of a system to avoid the formation of massive distrust in the user population after an incident. It is of course possible to introduce an additional operational principle highlighting the importance of building and maintaining trust. Since the rest of the book concentrates on other aspects of anti-fragile systems, it does not contain an explicit trust principle. However, the building of trust should permeate through all work done to create and operate anti-fragile systems, because the loss of trust is an inherent and general threat to all ICT systems that can cause user populations to abandon systems altogether.

What to learn from Part I

Part I modeled large ICT systems as complex adaptive systems and explained that positive feedback loops cause extreme global behavior with an intolerable impact. A complex system is fragile, robust, or anti-fragile to a particular class of negative impacts. It is not enough to create a complex system that is robust to a type of impact when the system is new. Because a complex system and its environment change over time, a robust system becomes fragile. While risk management methods can detect and mitigate many negative events, a complex system has too many interactions between its units and modules for a risk analyst to predict all incidents. It is particularly difficult for a group of stakeholders to predict rare and large-impact incidents called gray swans. Even worse, black swans may exist that are totally unpredictable to all stakeholders in the group.

It is necessary to build complex ICT systems that fail early when the impacts are still small and to learn from the remaining small events how to maintain and improve the systems. Four design principles, namely, *modularity*, *weak links*, *redundancy*, and *diversity*, and one operational principle, *fail fast*, were introduced to provide anti-fragility to different types of impact. The common goal of the design principles is to prevent inevitable local failures from propagating into global failures. The goal of the operational principle is to quickly determine vulnerabilities and remove them before they can cause serious damage. Here, a vulnerability can be a flaw in the design, a bug in the implementation, or a mistake in the operation or management of a system.

Because there is no absolute guarantee that a systemic failure will never occur, an owner or operator of a complex ICT system must build and maintain a trust relationship with the customers, especially since it can be argued that trust is fragile and distrust is robust. If a company allows distrust to grow, for example, by relegating, ignoring, or attacking individuals pointing out system weaknesses, then the company may not survive a failure, especially when it is heavily reported in the press.